A PROCESSOR SPECIFIC NOTES

VDK for BLACKfin DSPs (ADSP-21535)

User and Supervisor Modes

The BLACKfin architecture makes a distinction between execution in user and supervisor modes. VDK supports these modes and switches back and forth as necessary. When writing applications using VDK, you must be aware of the processor mode at the time the kernel calls the functions that you implement.

All thread functions execute in user mode, including `Run()` (`RunFunction()`) and `ErrorHandler()` (`ErrorFunction()`). Conversely, all Interrupt Service Routines execute in supervisor mode. The implementation function for device drivers (their single entry point) may be called by the kernel in either user mode or supervisor mode depending on the purpose of the call. Specifically, when the initialization routine is called at system boot time (`kDD_Init`), when an ISR calls `VDK_ISR_ACTIVATE_DEVICE_DRIVER_()` (`kDD_Activate`), or when a thread calls `DeviceIOCtl()` (`kDD_IOCtl`), the function executes in supervisor mode and within a critical region. For calls to `OpenDevice()` (`kDD_Open`), `CloseDevice()` (`kDD_Close`), `SyncRead()` (`kDD_SyncRead`), and `SyncWrite()` (`kDD_SyncWrite`), the function executes in user mode.

This limits somewhat the operations that can be performed in threads, but enforces the intended architecture model. System mode registers should be accessed at boot time by defining a device driver with the proper initialization steps and at run time through the `DeviceIOCtl()` function or from an

ISR. This model separates the algorithm code maintained in threads from hardware dependent details contained in device drivers and ISRs.

Critical and Unscheduled Regions

Note that, on the BLACKfin architecture, interrupts may only be enabled or disabled while in supervisor mode. The API functions `PushCriticalRegion()` and `PopCriticalRegion()` handle this mode change in a manner transparent to threads; however, entering and exiting critical regions takes approximately four times longer than entering and exiting unscheduled regions because of this overhead.

Exceptions

The BLACKfin architecture requires the use of "service exceptions" to enter supervisor mode. Therefore, VDK reserves service exception ID 0 (`EXCPT 0;`) for internal use. Additionally, the IDDE automatically generates a source file for all BLACKfin VDK projects that defines an entry point for any service or error exceptions you wish to trap. When an exception occurs, the VDK will intercept the exception, and if it is not the VDK exception, the user defined exception handler executes.

You should not manipulate the IMASK system register from within your exception handler. If you need to mask or unmask an interrupt in response to an exception, raise an interrupt and change the value of IMASK in the ISR after the exception handler.

ISR APIs

The BLACKfin assembly syntax requires the use of separate API macros depending on whether the arguments are constants (immediate values, enumerations) or data registers (`R0` through `R7`). The arguments to the default APIs described in the reference chapter must be constants. When

passing data registers as arguments, append "REG_" to the macro name as follows:

```
VDK_ISR_POST_SEMAPHORE_REG_(semaphore_num_);
VDK_ISR_ACTIVATE_DEVICE_REG_(dev_num_);
VDK_ISR_SET_EVENTBIT_REG_(eventbit_num_);
VDK_ISR_CLEAR_EVENTBIT_REG_(eventbit_num_);
VDK_ISR_LOG_HISTORY_REG_(enum_, value_, threadID_);
```

The API macros as defined in the reference guide (without "REG_") only accept constants as arguments. Passing a register name results in an assembler error.

Interrupts

The following hardware events (interrupts) are reserved for use by the VDK on the ADSP-21535:

- EVT_EVSW (a.k.a. EVT_EVX) — the software exception handler. User code handles software exceptions by modifying the source file created by the IDDE called ExceptionHandler-21535.asm.
- EVT_TMR — the interrupt associated with the timer integral to the processor core. This timer generates the interrupts for system ticks, and provides all VDK timing services. Disabling this timer will stop services such as sleeping, round robin scheduling, pending with timeout, periodic semaphores, etc.
- EVT_IVG15 — general interrupt #15. This interrupt is reserved for use by the scheduler and may not be used by the user in any manner.

The ADSP-21535 designates hardware events seven (IVG7) through thirteen (IVG13) as "general interrupts" and each maps to more than one physical peripheral. The IDDE generates source code templates with a single entry point per interrupt level rather than for each peripheral. Therefore, you are responsible for dispatching interrupts when more than one is used peripheral at the same level. The technique used depends on

the application, but typically, either chaining or a jump table is used. When chaining, a handler will check to see if the interrupt was caused by the specific peripheral it knows how to handle. If not, execution is passed to the next handler in the chain. A jump table makes use of an ID or constant to index into a table of function pointers to find the appropriate interrupt handler.

Timer

VDK Ticks are derived from the timer implemented in the inner processor core of the ADSP-21535 and are synchronized to the main core clock `CCLK`. However, this timer is disabled when the ADSP-21535 enters low power mode. Thus, all VDK timing services such as sleeping, time-outs, and periodic semaphores do not operate while the core is in IDLE or low power mode.

The ADSP-21535 simulator shipped with VisualDSP++ 2.0 exhibits an anomaly in the manner in which the timer is clocked. The simulator decrements the time once per instruction without taking into account the number of cycles required to dispatch the instruction. Thus, the period of a Tick will be a larger number of cycles than on the actual hardware. This behavior differs from execution in hardware.

Memory

By default, the VDK LDF file places all user code and data into a single section called “program.” Refer to the *Linker and Utilities Manual* for information on segmenting your code.

By default, the scratch SRAM is not used or defined in the LDF file and is not configured to be used as level one cache. If you choose to use the processor in a L1 cache mode, you should write an initialization device driver for configuration as described in “Device Drivers” on page 4-34.

VDK FOR ADSP-219x DSPs (ADSP-2191and ADSP-2192)

Interrupts

The following interrupts are reserved for use by the VDK on the ADSP-2191:

- `IRPTL[4]` — the timer interrupt. Timer 2 generates the interrupts for system ticks, and provides all VDK timing services. Disabling this timer will stop services such as sleeping, round robin scheduling, pending with timeout, periodic semaphores, etc. This interrupt is reserved for use by the scheduler and may not be used by the user in any manner.
- `IRPTL[15]` — general interrupt #15. This interrupt is reserved for use by the scheduler and may not be used by the user in any manner.

The following interrupts are reserved for use by the VDK on the ADSP-2192:

- `Timer[Bit 5/Priority 6]` — the timer interrupt. Timer 2 generates the interrupts for system ticks, and provides all VDK timing services. Disabling this timer will stop services such as sleeping, round robin scheduling, pending with timeout, periodic semaphores, etc. This interrupt is reserved for use by the scheduler and may not be used by the user in any manner.
- Reserved `ISR` — the reschedule interrupt. This interrupt is reserved for use by the scheduler and may not be used by the user in any manner. When using the VDK, there are some restrictions placed on the ISRs associated with the AC'97 codec port. Specifically, your entire ISR at priority level 16 (bit 15) must execute within a critical region.

You should never enable interrupts while servicing an AC'97 Frame, and a nested interrupt should never be allowed to occur once an AC'97 Frame ISR begins.

Software Mapped Interrupt Controller

The ADSP-2191 allows interrupt levels five (`IRPTL[5]`) through fourteen (`IRPTL[14]`) to be mapped in software to the on-chip peripherals (`IRPTL[4]` and `IRPTL[15]` are reserved by VDK). The IDDE generates source code templates with a single entry point per interrupt level rather than for each peripheral. Therefore, you are responsible for dispatching interrupts when more than one peripheral is used at the same level. The technique used depends on the application, but typically, either chaining or a jump table is used. When chaining, a handler will check to see if the interrupt was caused by the specific peripheral it knows how to handle. If not, execution is passed to the next handler in the chain. A jump table makes use of an ID or constant to index into a table of function pointers to find the appropriate interrupt handler. The ADSP-2192 maps peripherals to specific interrupt levels in hardware, so this initialization is not necessary.

VDK FOR ADSP-TS101

User and Supervisor Modes

The TigerSHARC architecture makes a distinction between execution in user and supervisor modes. VDK supports these modes and switches back and forth as necessary. When writing applications using VDK, you must be aware of the processor mode at the time the kernel calls the functions that you implement.

All thread functions execute in user mode, including `Run()` (`RunFunction()`) and `ErrorHandler()` (`ErrorFunction()`). Conversely, all interrupt service routines execute in supervisor mode. The implementation function for device drivers (their single entry point) may be called by the kernel in either user mode or supervisor mode depending on the purpose of the call. Specifically, when the initialization routine is called at system boot time (`kDD_Init`), when an ISR calls `VDK_ISR_ACTIVATE_DEVICE_DRIVER_()` (`kDD_Activate`), or when a thread calls `DeviceIOCtl()` (`kDD_IOCtl`), the function executes in supervisor mode and within a critical region. For calls from a thread to `OpenDevice()` (`kDD_Open`), `CloseDevice()` (`kDD_Close`), `SyncRead()` (`kDD_SyncRead`), and `SyncWrite()` (`kDD_SyncWrite`), the function executes in user mode.

This limits somewhat the operations that can be performed in threads, but enforces the intended architecture model. System mode registers should be initialized at boot time by defining a device driver with the proper initialization steps in its `kDD_Activate` portion. At runtime, access system registers through the `DeviceIOCtl()` function or from an ISR. This model separates the algorithm code maintained in threads from hardware dependent details contained in device drivers and ISRs.

Critical and Unscheduled Regions

In the TigerSHARC architecture, interrupts may only be enabled or disabled while in supervisor mode. The API functions PushCriticalRegion() and PopCriticalRegion() handle this mode change in a manner transparent to threads; however, entering and exiting critical regions takes considerably longer than entering and exiting unscheduled regions because of the overhead of entering supervisor mode.

Exceptions

The TigerSHARC architecture requires the use of a user software interrupt (a.k.a. service exception, trap) to enter supervisor mode. When you throw a software exception, the TRAP opcode incorporates a 5-bit identifier that is available as via a system status register (`SQSTAT[7:3]`) in the instruction handler.

VDK reserves exception `ID 0` (`TRAP 0;`) for internal use. Additionally, the IDDE automatically generates a source file for all TigerSHARC VDK projects that define an entry point for any service or error exceptions you wish to trap. When an exception occurs, the VDK intercepts the exception, and if it is not the VDK exception, the user-defined exception handler executes.

ISR APIs

The TigerSHARC assembly syntax requires the use of separate API macros depending on whether the arguments are constants (immediate values, enumerations) or compute block registers (`XR0` through `XR7`, or `YR0` through `YR7`). Note that `IALU` registers may not be used directly as arguments to ISR macros. The arguments to the default APIs described in the

"API Reference" on page 6-1 must be constants. When passing compute block registers as arguments, append "REG_" to the macro name as follows:

```
VDK_ISR_POST_SEMAPHORE_REG_(semaphore_num_)
VDK_ISR_ACTIVATE_DEVICE_REG_(dev_num_)
VDK_ISR_SET_EVENTBIT_REG_(eventbit_num_)
VDK_ISR_CLEAR_EVENTBIT_REG_(eventbit_num_)
VDK_ISR_LOG_HISTORY_REG_(enum_, value_, threadID_)
```

The API macros as defined in the API Reference chapter (without "REG_") only accepts constants as arguments. Passing a register name will result in an assembler error.

Interrupts

The following interrupts are reserved for use by the kernel on the ADSP-TS101:

- INT_TIMEROH (ILAT bit 52) — the high-priority interrupt associated with timer 0. This timer generates the interrupts for system ticks and provides all VDK timing services. Disabling this timer stops services, such as sleeping, round-robin scheduling, pending with timeout, periodic semaphores, etc.
- INT_TIMEROL (ILAT bit 2) — the low-priority interrupt associated with timer 0 is reserved for use by the scheduler and may not be used by the user in any manner.

All other interrupts are available for user applications and should be installed by right-clicking the **Interrupts** icon on the **VDK** tab of the **Project Window**.

Parallel Memory Accesses

The ADSP-TS101 takes advantage of the dual IALU architecture of TigerSHARC and allocates two stacks per thread. The number shown in the **VDK** tab of the **Project Window** specifies the number of 32-bit words per stack. Since each thread has two stacks (corresponding to the J and K IALUs), multiply this number by two to compute the number of memory words used per thread, or by eight for the number of bytes used per thread. See the *VisualDSP++ 2.0 C/C++ Compiler & Library Manual for TigerSHARC DSPs* for more information on the use of stacks and the run-time model maintained by C/C++ compiler.

The VDK performs parallel addressing by using multiple heaps. The default LDF file for VDK projects defines two heaps (each 16K words). These sections must appear with the names used in the default LDF file. You may define additional heaps using the multiple heap API. If you do so, you must define the heaps in an object file or library that is listed on the line in the LDF file that defines the symbol $LIBS after the library VDK-CORE-TS101.dlb and before the symbol $BASE_LIBRARIES. This ensures that the new heaps are not placed after the token that terminates the heap list. See the *VisualDSP++ 2.0 C/C++ Compiler & Library Manual for TigerSHARC DSPs* for more information on using multiple heaps.

VDK FOR ADSP-2106X and ADSP-2116X DSPs

Analog Devices ADSP-2106x family includes ADSP-21060, ADSP-21061, ADSP-21062, and ADSP-21065L processors, and ADSP-2116x family includes ADSP-21160 and ADSP-21161 processors.

Interrupts

The following interrupts are reserved for use by the VDK on the ADSP-2106x devices:

- `IRPTL[4]` — the high-priority timer interrupt (`TMZHI`). The timer (`Timer0` on ADSP-21065L) generates the interrupts for system ticks and provides all VDK timing services. Disabling this timer stops services, such as sleeping, round-robin scheduling, pending with timeout, periodic semaphores, etc. This interrupt is reserved for use by the scheduler and may not be used by the user in any manner.
- `IRPTL[31]` — user software interrupt #3. This interrupt is reserved for use by the scheduler and may not be used by the user in any manner.

On ADSP-21065L, `Timer1` is available for use by user code, but only the low-priority timer interrupt (`TMZLI`) is available for servicing it as the high-priority interrupt (`TMZHI`) is reserved by VDK for `Timer0`. By default, VDK assigns `Timer1` to the low-priority timer interrupt (`IRPTL[23]`), so using it only requires an interrupt handler to be defined for `TMZLI` and for `Timer1` to be initialized.

The following interrupts are reserved for use by the VDK on the ADSP-2116x devices:

- `IRPTL[4]` — the high-priority timer interrupt (`TMZHI`). The timer generates the interrupts for system ticks and provides all VDK timing services. Disabling this timer stops services, such as sleeping, round-robin scheduling, pending with timeout, periodic semaphores, etc. This interrupt is reserved for use by the scheduler and may not be used by the user in any manner.
- `IRPTL[30]` — user software interrupt #3. This interrupt is reserved for use by the scheduler and may not be used by the user in any manner.

Register Usage

The ADSP-21x6x devices provide a set of alternate (or background) registers for both the computation units and the DAGs. VDK does not save and restore these alternate registers during context switches, but instead employs them to accelerate entry to and exit from the scheduler interrupt by providing it with a dedicated C run-time environment. This means that the following alternate registers contain fixed values and must not be changed by user code:

Register	Value
`M5', M13'`	0
`M6', M14'`	1
`M7', M15'`	1
`B6', B7'`	System stack base
`L6', L7'`	System stack length
`L2'-L5', L10'-L15'`	0

Since these are used at interrupt level (which may be entered at any time), it is not sufficient to save and restore these registers after use, except within a critical region.

The following alternate registers are not used by VDK and are available for use by user code:

```
I0', M0', B0', L0',
I1', M1', B1', L1',
I8', M8', B8', L8',
I9', M9', B9', L9'
```

Note that these registers do not form part of the thread context and, hence, are not context-switched between threads. Therefore, their most appropriate use is in implementing fast I/O interrupt handlers for devices where DMA is not available.

The remaining alternate registers,

```
R0'-R15',
I2'-I7', I10'-I15',
M2'-M4', M10'-M12',
B2'-B5', B10'-B15',
```

are used by the Scheduler ISR and, hence, may change at any time unless inside a critical region. If they are to be used in a user-defined interrupt service routine, then their contents must be saved and restored before returning from the interrupt (as for the primary registers).

Note that the background multiplier result register, `MRB`, is not considered to be an alternate register in the above discussion, and that both `MRB` and `MRF` are context-switched between threads.